本系列决定改个名称，仍然沿用原来的序号。目的仍然是搜集与游戏物理相关的源码，将源码与公式联系起来。因为大多数论文或书籍都没附带代码，以我现在的技术自己写一份代码很困难并且容易跑偏，在互联网上寻找相关的代码又很困难。只好反着来，先看代码，然后再阅读相关的论文。

Christopher Batty是一位大神级别的人物，他有与流体相关五份相似的开源代码

这篇文章我找到的源码为https://github.com/christopherbatty/VariationalViscosity3D，它是对SCA 2008 paper "Accurate Viscous Free Surfaces[...]" by Batty & Bridson 的3D复现。这份源码虽然是c++的，但是运行和调试起来都很容易。这个库还有2D版本https://github.com/christopherbatty/VariationalViscosity2D。这位名叫的其它github库运行起来也都很容易。

我把源码分别翻译到了python和unity中，因此也获得了一些经验值。下面是unity中的3D效果：

https://www.cs.ubc.ca/labs/imager/tr/2007/Batty_VariationalFluids/

写python主要是能够方便地查看变量与矩阵的数值。

Air构建

如果能把项目运行起来，那么学习难度瞬间下降99%。不过能够熟练地运行起别人的工程也花了我很久时间。我在windows10上运行起这个工程的方法如下：

首先打开2D版本的源码文件夹，发现里面既没有.vxproj文件，也没有CMakeLists.txt。于是我用VisualStudio2019新建一个工程，把2D版本的源码弄进去，点击运行一遍，报了几个错。可以说是打怪前的预热活动了。

第一个错是不安全的函数，_CRT_SECURE_NO_WARNINGS，解决这个问题的方法stackoverflow和csdn上有一大堆。轻松秒掉。第二个错是函数重定义，我粗暴地把重定义的函数删掉了，同样轻松秒掉。

第三个错是找不到GL/glut.h。这个错误足够干掉一大堆萌新了。并且stackoverflow和csdn上的答案多半不靠谱。借此重新整理一下这个问题的解决方法。

glut库是一个20多年前就停止更新了的库，现在一般用freeglut库，可以实现与glut库相同的功能。首先去freeglut官网下载源码，解压打开，这样就得到了需要的头文件。

文件夹里面有个CMakeLists.txt，这意味着又得去下载个CMake。不过我一般习惯用命令行而不是gui使用CMake。也就是在有cmakeLists.txt的文件夹路径处输入cmd，回车，然后cmake CMakeLists.txt这样。

成功生成项目工程后，用VS打开ALL_BUILD.vxproj，然后运行，运行成功后会报错无法打开ALL_BUILD，这样就行了。这意味我们需要的lib和dll也生成了。然后回到原工程，配置属性-VC++目录-包含目录，把freeglut\include所在路径放进去，然后链接器-输入，把freeglutd.lib所在路径加文件名放进去，然后把freeglutd.dll扔到system32里去，这样就完成了。顺便感叹一句，everything这个软件的复制完整路径和文件名在为c++项目添加.lib的时候真好用。

第四个错是无法解析的符号，这类错误经常出现。而这个库中出现的主要原因是，我使用了64位的freeglutd.lib，而本项目的调试器x86的。所以很简单地把调试器改成Debugx64就行了。

Air源码解读

第一步：初始化

这些源码中，求解区域内也有障碍物，而定义这些障碍物的方式用的是距离场。首先算出到所有障碍物中心的最小距离

```
float circle_phi(const vec2f& position, const vec2f& centre, float
radius) {
    return (dist(position, centre) - radius);
}

float boundary_phi(const vec2f& position) {
    float phi0 = -circle_phi(position, c0, rad0);
    float phi1 = circle_phi(position, c1, rad1);
    float phi2 = circle_phi(position, c2, rad2);
    float phi3 = circle_phi(position, c3, rad3);
    return min(min(phi0, phi1), min(phi2, phi3));
}
```

当然，如果第一次见到这样写的代码肯定还是很懵逼。所以可以先去看看水平集LevelSet相关的代码，以及ld大神有关RayMarch的文章，里面距离场用得相当多。这些距离场被存进一个叫nodal_solid_phi的数组中，负数代表在障碍物里面，正数代表在障碍物外面。

第二步：对流粒子

然后就可以进入主循环FluidSim::advance了。其中第一个函数advect_particles，首先让粒子用二阶RungeKutta的方式移动。不过这里的粒子自己本身并不包含速度这个属性，需要速度的时候直接采样网格上的速度。计算速度时这份源码采用的是线性插值，但也可以改成GAMES201里介绍的spline插值，具体可修改如下：

RungeKutta嘛，就是泰勒展开变个形状罢了。在FluidSim::trace_rk2，可以把它修改成三四阶的，后两者倒是经常能在别的代码里看见。

在advect_particles函数里，粒子移动完后，还会判断它是否在障碍物里面。判断方法同样是线性插值

```
float phi_value = interpolate_value(particles[p]/dx,
nodal_solid_phi);
```

与前面说的一样，如果这个值是负的，那么就是在障碍物里面，就要把位置修正，减去斜率乘以距离场的值。

也可以这么说，求解域是个凹凸不平的表面，高度小于零的地方就是坑，也就是障碍物。如果粒子在高度小于零的地方，那么说明它掉坑里去了，得把它拉回来。

第二步：对流网格速度

这份源码中的粒子是不携带速度的，所以要更新速度，仍然需要在网格上进行。这份源码用的是半隐式 + 二阶RungeKutta。RungeKutta可以改成三阶四阶的，插值方式可以改为三线性插值，与之前改法一样，半隐式可以改为欧拉中点，或者BEFCC。

第三步：压力泊松方程

然后就是这份源码，或者说是本篇中最精彩的部分，计算压力泊松方程了。

这儿的网格用的是交错网格，用类似于有限体积法的方式计算速度的权重。越靠近障碍物的速度，其权重越小，最大为一，最小为零。交错网格分配如下

关于泊松压力方程基本介绍我已经在第十三篇里写了，这里仅写一些变换的。泊松压力方程的形式如下：

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$$

其中p是压力，u是X轴方向上的速度，v是Y轴方向上的速度。我所见过的源码中，99%用二阶精度的中心差分来离散上面这个方程。离散成如下形式

$$\frac{p_{i-1,j} + p_{i-1,j} - 2p_{i,j}}{\Delta x^2} + \frac{p_{i,j-1} + p_{i,j+1} - 2p_{i,j}}{\Delta y^2} = divergence$$

可以把它当成矩阵来解，化成下面这样的形式。

$$Ax = b$$

上面x就是待求解的压力，b就是散度，A就是中心差分的系数矩阵。假如网格是4x4的话，那么矩阵A就有4x4 = 16列以及同样的行数。x和b就是16行单列的向量了。系数矩阵规则如下，假如只有边界是障碍物的话，假设网格长度为dx，那么如果自身上下左右某个相邻网格不是障碍物的话，就给它加上系数-1/(dx^2)，而自身加上系数1/(dx^2)。如果dx是1的话，那么A矩阵看起来如下

$$A = \begin{bmatrix}
2 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 3 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 2
\end{bmatrix}$$

而这份源码再让中心差分系数乘上速度权重，让靠近障碍物的速度权重变小，最后解完泊松压力方程，那么障碍物附近算出来的压力必将变大，从而将粒子尽量推离障碍物。

```
float term = u_weights(i+1,j) * dt / sqr(dx);
matrix.add_to_element(index, index, term);
matrix.add_to_element(index, index + 1, -term);
rhs[index] -= u_weights(i+1,j)*u(i+1,j) / dx;
```

比如上面FluidSim::solve_pressure里的这几行，就是根据压力网格(i,j)的右边离障碍物的远近，来设置差分系数。散度计算也在最后一行做了。

不过看看上面那个系数矩阵，大部分元素都是零，因此可以使用稀疏矩阵来减少内存消耗。所谓稀疏矩阵，简单来说就是两个数组，一个存索引，一个存数据，并且只存储非零数据。不过现在，由于python上和GPU上的动态数组并不是很好用，我们可以使用与Batty的源码有很大区别的另一种稀疏矩阵形式。

由于我们是用中心差分解泊松压力方程，二维情况下每行最多5个非零元素，分别是自己以及上下左右邻居。三维情况下每行最多7个元素，分部是自己以及上下左右前后。所以我们可以把(nx * ny)行(nx * ny)列 的矩阵减少成(nx * ny) 行 5 列的矩阵。

而且我们现在没用预处理子，也不实现矩阵分解，我们连这个矩阵都可以完全不用存了。回想一下jacobi方法和GuassSeidel方法，压根没存过矩阵，只需要在迭代的时候把系数乘上去就行了。另一个很好的例子是taichi库中的cg_poisson.py，在没有存储矩阵的情况下完成了共轭梯度法解压力泊松方程。我用numpy重写一遍的代码在

不过，现在我们还是试试这种(nx * ny)行5列的矩阵，之后对其进行矩阵分解以及预处理更加方便一些，当然把它拆成五个数组也可以。

组装完矩阵，接下来就应该求解了。之前使用的Jacobi和GaussSidele 效率实在过于低下，如果流体效果看起来不对，多半是这两种迭代方法的锅。这份源码中使用了不完全CholeSky预处理共轭梯度法(incomplete Cholesky Preconditioned Conjugate Gradient solver)求解压力泊松方程。

老人看手机.jpg。

这方法有些复杂，我计划在下一篇中解释。现在我们可以用个简化的，只用共轭梯度就行了，效果也不会很差。

共轭梯度，要想误差r^Tr到达1e-10一下，4096个网格也仅仅需要不到100次循环。如果换成Jacobi或者GaussSeidel，循环需要上千次甚至上万次。

最后一步矫正速度

首先是FluidSim::solve_pressure中最后一部分让速度减去压力梯度，继而让速度符合不可压缩的性质。

然后FluidSim::extrapolate，作者的解释是，由于半隐式对流网格速度时，可能向所有的方向插值。万一插值的地方在障碍物里，那么速度就变成零了。

【这不河里】

所以还得给障碍物里靠近边界的地方平均一些速度下来，而障碍物外部的速度不受到影响。

如果障碍物也有速度的话，那么这些障碍物里的速度还得根据障碍物的法向量做调整。这就是FluidSim::constrain_velocity的作用。

虽然这篇论文标题写的是流固耦合，但给出的源码中并没有流固耦合的实现。并且直接搜索Solid Fluid Coupling搜不出太多代码。此时搜索的关键字应该为"Immersed Boundary Method"。

第二关：Fluid源码解读

与Air源码最大的不同，这份源码有两套距离场系统。第一套是之前的障碍物，第二套则是流体自身的距离场liquid_phi。粒子本身还有半径，比网格长度小一些。

```
particle_radius = dx/sqrt(2.0f);
```

第三关：Fluid3D

我放弃了，实在看不懂粘性矩阵来搞什么

我日，GPU上算共轭梯度，点积怎么算啊，并行算法不能对同一处内存同时操作啊。卡在这里了。

我心情耗尽了，三维和二维根本不是一个概念，重新换了。

Dual Contour 看

https://www.boristhebrave.com/2018/04/15/dual-contouring-tutorial/

正在学习CUDA中，计划在CUDA上实现一些共轭梯度之类的算法，本篇教程先搁置了。直接抄NVIDIA库的就行了

构建

rigidCoupling这份源码自带.sln，双击打开并运行，又蹦出来一个缺少依赖的错误Eigen/Sparse。那么去网上下载一份Eigen即可。Eigen是广受欢迎的数值算法库，比如Cholesky分解，SVD分解等都能在这里找到代码实现。然后把带cmakeLists.txt的文件路径放到引用目录中即可。